PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/445, 3/14 — A1

(11) International Publication Number: WO 95/15524

(43) International Publication Date: 8 June 1995 (08.06.95)

(21) International Application Number: PCT/US94/00016

(22) International Filing Date: 3 January 1994 (03.01.94)

(30) Priority Data:
08/161,384 2 December 1993 (02.12.93) US

(71) Applicant: TALIGENT, INC. [US/US]; 10201 N. de Anza Boulevard, Cupertino, CA 95014 (US).

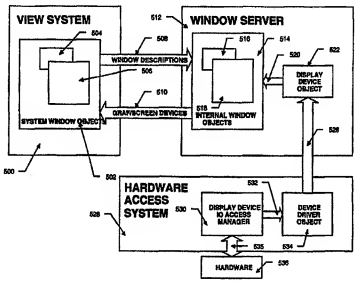
(72) Inventors: ZIAS, Jeff, A.; 2616 Estella Drive, Santa Clara, CA 95051 (US). MARSH, Donald, M.; 476 Dell Avenue, Mountain View, CA 94043 (US).

(74) Agent: STEPHENS, Keith; Taligent, Inc., 10201 N. De Anza Boulevard, Cupertino, CA 95014 (US).

(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR DISPLAYING HARDWARE DEPENDENT GRAPHICS IN AN OBJECT-ORIENTED OPERATING SYSTEM



(57) Abstract

Screen objects used by the application programs to draw or paint on the display screen are created in accordance with a predefined class structure which represents a generic display system. Developers of specialized hardware develop specialized classes based on the predefined structure to handle specific command sets and protocols. When the system is initially powered up or reconfigured, the screen objects are created from the generic and specialized classes as necessary by examining the actual display hardware present in the system. When the specialized classes are used to create the screen objects, the screen objects receive the specialized command sets and protocols necessary to interact with display hardware which have been provided by the hardware developers.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgyzstan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

METHOD AND APPARATUS FOR DISPLAYING HARDWARE DEPENDENT GRAPHICS IN AN OBJECT-ORIENTED OPERATING SYSTEM

COPYRIGHT NOTIFICATION

Field of the Invention

This invention relates generally to improvements in computer systems and, more particularly, to operating system software for managing hardware dependent graphics in a windows-oriented graphical user interface.

Background of the Invention

One of the most important aspects of a modern computing system is the interface between the human user and the machine. The earliest and most popular type of interface was text based; a user communicated with the machine by typing text characters on a keyboard and the machine communicated with the user by displaying text characters on a display screen. More recently, graphic user interfaces have become popular in which the machine communicates with the user by displaying graphics, including text and pictures, on a display screen and the user communicates with the machine both by typing in text-based commands and by manipulating the displayed pictures with the pointing device, such as a mouse.

Many modern computer systems operate with a graphic user interface called a "window environment". In a typical window environment, the graphical display portrayed by the display screen is arranged to resemble the surface of an electronic "desktop" and each application program running on the computer is represented as one or more electronic "paper sheets" displayed in rectangular regions of the screen called "windows".

Each rectangular region generally displays information which is generated by the associated application program and there may be several window regions simultaneously present on the desktop with each window region representing information generated by different application programs. An application program presents information to the user through each window by drawing or "painting" images, graphics or text within the window region. The user, in turn, communicates with the application both by "pointing" at objects in the window region with a cursor which is controlled by a pointing device and manipulating or moving the objects and also by typing information into the keyboard. The window

regions may also be moved around on the display screen and changed in size and appearance so that the user can arrange the desktop in a convenient manner.

In general, the window environment described above is part of the computer operating system. The operating system also typically includes a collection of utility programs that enable the computer system to perform basic operations, such as, storing and retrieving information on a disk memory, and performing file operations including the creating, naming and renaming of files and, in some cases, performing diagnostic operations in order to discover or recover from malfunctions.

The last part of the computing system is the "application program" which interacts with the operating system to provide much higher level functionality, perform a specific task and provide a direct interface with the user. The application program typically makes use of the operating system functions by sending out a series of tasks and commands to the operating system which then performs the required task. For example, the application may request that the operating system store particular information on the computer disk memory or display information on the video display.

In a graphically-oriented environment, such as the windowing environment described above, the display capabilities of the video display device are crucial to the operation of the system. Therefore, it is important that both the application program and the operating system operate efficiently with the display hardware which is present in the system. In particular, due to the large volume of graphic information, the display hardware may have special accelerators or video circuitry built in. The display hardware may also have specialized capabilities including circuitry for performing such video effects as wipes, chroma keying, mosaics, motion video and other effects which would be difficult, if not impossible, to accomplish with software. Consequently, the various commands and protocols required by this specialized hardware must be made available to the application programs and, ultimately, the user.

However, the provision of specific commands and protocols for use with specialized hardware is practically difficult because a given application program must be able to operate on computer systems which are configured in many different ways and may or may not have specialized display hardware. Further, video display hardware that is manufactured by different manufacturers may have completely different protocols. If the capabilities for communicating directly with the specialized hardware are provided directly in the application program, inefficient program development time results.

For example, Figure 1 is schematic illustration of a typical computer system utilizing both an application program and an operating system. The computer

system is schematically illustrated by dotted box 100, the application program is represented by box 102 and the operating system by box 106 and the interaction between the application program 102 and the operating system 106 is illustrated schematically by arrow 104. This division between an application program and an operating system is used in common on many types of computer systems ranging from main frames to personal computers.

The method for handling graphical displays, however, varies from computer to computer, and, in this regard, Figure 1 is more representative of a prior art personal computer system. In order to provide a graphical display, the application program 102 interacts (as shown schematically by arrow 108) with display driver software 110. In the arrangement shown in Figure 1, display driver software 110 is a separate program which is provided with application program 102. Display driver software 102 receives display commands from the application program and generates the signals necessary to control the display hardware. As each type of display hardware has its own particular format and control commands, the display driver software 110 generally must be specifically designed to operate with one type of display.

The display driver software 110 generates an information stream containing the display information and associated commands as shown schematically as arrow 114. The display information stream is, in turn, applied to a display adapter 112 which contains circuitry that converts the information stream into electrical signals and the signals are, in turn, sent over a cable 116 to a display 118. The display 118 typically contains circuits which generate a two-dimensional "raster" on the screen and the incoming display information screen is used to modify points or "pixels" on the raster necessary to develop the actual graphical display. The result is a two-dimensional graphical display which is generally in color.

The configuration shown in Figure 1 has the advantage that the display driver software is provided with the application program and, thus, the application program developer can write display drivers for specific display hardware, utilizing all of the available capabilities. The disadvantage with the Figure 1 arrangement became apparent as the number of specialized display hardware types and display types proliferated. Since, in general, separate driver software had to be written for each hardware type, the number of display drivers which were required for each application program so that the program was compatible with, and could take advantage of, all of the hardware that was available increased as well. Application program developers had to provide many display drivers with each application program, resulting in wasted time and effort and wasted disk space to hold the many drivers of which only one or two were of interest to any particular user. Unfortunately, if a user purchased an application program and it did not include a

display driver which would control the display which the user owned, at best, the application program was unable to exploit available display capabilities. More likely, unpredictable operation occurred and either the program did operate satisfactorily or did not operate at all with the display.

In order to overcome the aforementioned difficulties, the prior art arrangement was modified as shown in Figure 2. In this new arrangement, computer system 200 is still controlled by application program 202 which cooperates, as shown schematically by arrow 204, with operating system 206. However, in the system shown in Figure 2 operating system 206 includes a "view system" 208 which manages the graphical interaction with the application program 202. Operating system 206 also includes the display drivers 214. In order to use specialized hardware capabilities, a separate display driver still must be provided for each different type of display hardware, but the display drivers 214 are now sold with, and part of, the operating system 206. Consequently, it is not necessary for each application program to have its own set of display drivers.

More particularly, an application program, such as application program 202 communicates with the display driver 214 by means of a standardized interface 208 which is called a "view system". In this type of system, application program 202 displays information on the display 222 by writing into a screen buffer which is part of the view system 208. The view system 208, in turn, forwards display information, as shown by arrow 212, to display driver 214 which, as previously described, converts the information into the format required by the particular display device. The output of the display driver 214 as provided (illustratively shown by arrow 216) to display adapter 218 where it is converted into electrical signals that are transmitted, via cable 220, to the display 222.

The configuration shown in Figure 2 has the advantage that the application program developer need not worry about the specific computer and display combinations on which the program is to ultimately run. However, in order to provide advanced display capabilities to the application program, the operating system now must include specific drivers for each display hardware type and must somehow transmit the specialized commands and protocols required by these drivers back through the view system to the application program. Therefore, the view system must either contain command sets for specialized hardware, most of which will not be used by more than a small fraction of the operating system users, or must be able to transmit the command sets from the drivers to the application program. Furthermore, as new types of display hardware are being continually developed, the operating system will be continually "out-of-date" and will not be able to provide capabilities for newly-developed hardware without providing new display drivers in subsequent "versions".

Accordingly, it is an object of the present invention to provide an object-oriented operating system which can dynamically determine and utilize the hardware capabilities of the display devices attached to the computer system.

It is another object of the present invention to dynamically generate display objects that incorporate specialized hardware capabilities and which can be manipulated under control of application program to control the specialized hardware.

Summary of the Invention

The foregoing problems are overcome and the foregoing objects are achieved in one illustrative embodiment of the invention in which screen objects used by the application programs to draw or paint on the display screen are created in accordance with a predefined class structure which represents a generic display system. Developers of specialized hardware develop specialized classes based on the predefined structure to handle specific command sets and protocols.

When the system is initially powered up or reconfigured, the screen objects are created from the generic and specialized classes as necessary by examining the actual display hardware present in the system. When the specialized classes are used to create the screen objects, the screen objects receive the specialized command sets and protocols necessary to interact with display hardware which have been provided by the hardware developers.

More particularly, specialized display driver capabilities are constructed dynamically by creating screen objects from a generic and specialized class mix that is selected using specifications obtained from information perataining to the specific display device.

Brief Description of the Drawings

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings, in which:

Figure 1 is a schematic block diagram of a prior art computer system showing the relationship of the application program to the operating system and the display drivers.

Figure 2 is a schematic block diagram of a modification of the prior art system shown in Figure 1 to allow the application program to interface with a standard graphical interface.

Figure 3 is a block schematic diagram of a computer system, for example, a personal computer system on which the inventive object oriented operating system operates.

Figure 4 is a simplified schematic block diagram of an application program and the main components of a windows-oriented operating system constructed in accordance with the principles of the present invention.

Figure 5 is a block schematic diagram illustrating in more detail the main components of a windows-oriented operating system including the view system, the window server and, the hardware access system.

Figure 6 is a simplified class hierarchy diagram illustrating the base class, subclasses and associated classes involved in the construction of a specialized device driver capable of displaying motion video.

Figure 7 is a simplified class hierarchy diagram illustrating the base class, subclasses and associated classes involved in the construction of a specialized screen frame buffer object which is capable of displaying motion video.

Figure 8 is an illustrative flow chart of a method by which a specialized display driver object for a video hardware card is created.

Figure 9 is a schematic illustration of two video hardware devices both of which are connected to a system input/output bus illustrating the retrieval of configuration parameters from onboard ROM memories.

Figure 10 is an illustrative flow chart of a method by which an application program obtains specialized screen buffer objects incorporating the functionality required to draw on a specialized graphical display device.

Detailed Description of a Preferred Embodiment of the Invention

The invention is preferably practiced in the context of an operating system resident on a personal computer such as the IBM® PS/2® or Apple® Macintosh® computer. A representative hardware environment is depicted in Figure 3, which illustrates a typical hardware configuration of a computer 300 in accordance with the subject invention. The computer 300 is controlled by a central processing unit 302 (which may be a conventional microprocessor) and a number of other units, all interconnected via a system bus 308, are provided to accomplish specific tasks. Although a particular computer may only have some of the units illustrated in Figure 3, or may have additional components not shown, most computers will include at least the units shown.

Specifically, computer 300 shown in Figure 3 includes a random access memory (RAM) 306 for temporary storage of information, a read only memory (ROM) 304 for permanent storage of the computer's configuration and basic operating commands and an input/output (I/O) adapter 310 for connecting peripheral devices such as a disk unit 313 and printer 314 to the bus 308, via cables 315 and 312, respectively. A user interface adapter 316 is also provided for connecting input devices, such as a keyboard 320, and other known interface devices

including mice, speakers and microphones to the bus 308. Visual output is provided by a display adapter 318 which connects the bus 308 to a display device 322, such as a video monitor. The workstation has resident thereon and is controlled and coordinated by operating system software such as the Apple System/7, operating system.

In a preferred embodiment, the invention is implemented in the C++ programming language using object-oriented programming techniques. C++ is a compiled language, that is, programs are written in a human-readable script and this script is then provided to another program called a compiler which generates a machine-readable numeric code that can be loaded into, and directly executed by, a computer. As described below, the C++ language has certain characteristics which allow a software developer to easily use programs written by others while still providing a great deal of control over the reuse of programs to prevent their destruction or improper use. The C++ language is well-known and many articles and texts are available which describe the language in detail. In addition, C++ compilers are commercially available from several vendors including Borland International, Inc. and Microsoft Corporation. Accordingly, for reasons of clarity, the details of the C++ language and the operation of the C++ compiler will not be discussed further in detail herein.

As will be understood by those skilled in the art, Object-Oriented Programming (OOP) techniques involve the definition, creation, use and destruction of "objects". These objects are software entities comprising data elements and routines, or functions, which manipulate the data elements. The data and related functions are treated by the software as an entity and can be created, used and deleted as if they were a single item. Together, the data and functions enable objects to model virtually any real-world entity in terms of its characteristics, which can be represented by the data elements, and its behavior, which can be represented by its data manipulation functions. In this way, objects can model concrete things like people and computers, and they can also model abstract concepts like numbers or geometrical designs.

Objects are defined by creating "classes" which are not objects themselves, but which act as templates that instruct the compiler how to construct the actual object. A class may, for example, specify the number and type of data variables and the steps involved in the functions which manipulate the data. An object is actually created in the program by means of a special function called a constructor which uses the corresponding class definition and additional information, such as arguments provided during object creation, to construct the object. Likewise objects are destroyed by a special function called a destructor. Objects may be used by using their data and invoking their functions.

The principle benefits of object-oriented programming techniques arise out of three basic principles; encapsulation, polymorphism and inheritance. More specifically, objects can be designed to hide, or encapsulate, all, or a portion of, the internal data structure and the internal functions. More particularly, during program design, a program developer can define objects in which all or some of the data variables and all or some of the related functions are considered "private" or for use only by the object itself. Other data or functions can be declared "public" or available for use by other programs. Access to the private variables by other programs can be controlled by defining public functions for an object which access the object's private data. The public functions form a controlled and consistent interface between the private data and the "outside" world. Any attempt to write program code which directly accesses the private variables causes the compiler to generate an error during program compilation which error stops the compilation process and prevents the program from being run.

Polymorphism is a concept which allows objects and functions which have the same overall format, but which work with different data, to function differently in order to produce consistent results. For example, an addition function may be defined as variable A plus variable B (A+B) and this same format can be used whether the A and B are numbers, characters or dollars and cents. However, the actual program code which performs the addition may differ widely depending on the type of variables that comprise A and B. Polymorphism allows three separate function definitions to be written, one for each type of variable (numbers, characters and dollars). After the functions have been defined, a program can later refer to the addition function by its common format (A+B) and, during compilation, the C++ compiler will determine which of the three functions is actually being used by examining the variable types. The compiler will then substitute the proper function code. Polymorphism allows similar functions which produce analogous results to be "grouped" in the program source code to produce a more logical and clear program flow.

The third principle which underlies object-oriented programming is inheritance, which allows program developers to easily reuse pre-existing programs and to avoid creating software from scratch. The principle of inheritance allows a software developer to declare classes (and the objects which are later created from them) as related. Specifically, classes may be designated as subclasses of other base classes. A subclass "inherits" and has access to all of the public functions of its base classes just as if these function appeared in the subclass. Alternatively, a subclass can override some or all of its inherited functions or may modify some or all of its inherited functions merely by defining a new function with the same form (overriding or modification does not alter the function in the base class, but merely

modifies the use of the function in the subclass). The creation of a new subclass which has some of the functionality (with selective modification) of another class allows software developers to easily customize existing code to meet their particular needs.

Although object-oriented programming offers significant improvements over other programming concepts, program development still requires significant outlays of time and effort, especially if no pre-existing software programs are available for modification. Consequently, a prior art approach has been to provide a program developer with a set of pre-defined, interconnected classes which create a set of objects and additional miscellaneous routines that are all directed to performing commonly-encountered tasks in a particular environment. Such pre-defined classes and libraries are typically called "application frameworks" and essentially provide a pre-fabricated structure for a working application.

For example, an application framework for a user interface might provide a set of pre-defined graphic interface objects which create windows, scroll bars, menus, etc. and provide the support and "default" behavior for these graphic interface objects. Since application frameworks are based on object-oriented techniques, the pre-defined classes can be used as base classes and the built-in default behavior can be inherited by developer-defined subclasses and either modified or overridden to allow developers to extend the framework and create customized solutions in a particular area of expertise. This object-oriented approach provides a major advantage over traditional programming since the programmer is not changing the original program, but rather extending the capabilities of the original program. In addition, developers are not blindly working through layers of code because the framework provides architectural guidance and modeling and, at the same time, frees the developers to supply specific actions unique to the problem domain.

There are many kinds of application frameworks available, depending on the level of the system involved and the kind of problem to be solved. The types of frameworks range from high-level application frameworks that assist in developing a user interface, to lower-level frameworks that provide basic system software services such as communications, printing, file systems support, graphics, etc. Commercial examples of application frameworks include MacApp (Apple), Bedrock (Symantec), OWL (Borland), NeXT Step App Kit (NeXT), and Smalltalk-80 MVC (ParcPlace).

While the application framework approach utilizes all the principles of encapsulation, polymorphism, and inheritance in the object layer, and is a substantial improvement over other programming techniques, there are difficulties which arise. These difficulties are caused by the fact that it is easy for developers to reuse their own objects, but it is difficult for the developers to use objects generated

by other programs. Further, application frameworks generally consist of one or more object "layers" on top of a monolithic operating system and even with the flexibility of the object layer, it is still often necessary to directly interact with the underlying operating system by means of awkward procedural calls.

In the same way that an application framework provides the developer with prefab functionality for an application program, a system framework, such as that included in a preferred embodiment, can provide a prefab functionality for system level services which developers can modify or override to create customized solutions, thereby avoiding the awkward procedural calls necessary with the prior art application frameworks programs. For example, consider a display framework which could provide the foundation for creating, deleting and manipulating windows to display information generated by an application program. An application software developer who needed these capabilities would ordinarily have to write specific routines to provide them. To do this with a framework, the developer only needs to supply the characteristics and behavior of the finished display, while the framework provides the actual routines which perform the tasks.

A preferred embodiment takes the concept of frameworks and applies it throughout the entire system, including the application and the operating system. For the commercial or corporate developer, systems integrator, or OEM, this means all of the advantages that have been illustrated for a framework such as MacApp can be leveraged not only at the application level for such things as text and user interfaces, but also at the system level, for services such as printing, graphics, multimedia, file systems, I/O, testing, etc.

Fig. 4 shows a schematic overview of an application program and a portion of an operating system program which is specialized for a windows-oriented environment. The operating system is shown as dotted box 402 and generally consists of a view system 404, a graphic system 406, a window server 412, a hardware configuration system 416 and a hardware access system 418. In an object-oriented operating system, each of the systems represented by boxes 404-418 may, for example, be a separate object or a collection of objects. There may be several application programs running simultaneously in the system of which only program 400 is shown for clarity.

In order to display information, application program 400 interfaces (as illustrated schematically by arrow 408) with view system 404 and graphic system 406 generally by creating and interfacing with various objects as will hereinafter be described in detail. View system 404 manages the window areas associated with application 400 and contains objects that are capable of creating, destroying and repainting windows. The view system also maintains screen buffer objects which

are controlled by application program 400 to actually write or paint on the display devices in the system.

View system 400 includes a subsystem, 406, called a graphic system with which it communicates shown schematically by arrow 409. The graphic subsystem contains objects which in turn facilitate the drawing of graphical images such as lines, circles, squares, etc. on the display screen.

In a windows-oriented environment, view system 404 communicates with another main component called a window server 412, which communication is shown schematically by arrow 410. The main function of window server 412 is to divide the available screen area among all of the running application programs; the window server acts essentially as a system level window manager. More specifically, view system 404 creates windows that have certain extents or bounds, and, based on the ordering of the windows, and the extent of the window overlaps, window server 412 can determine the visible area of each window. This visible area is communicated back to each application program, such as application program 400 and all application programs are expected to restrict drawing to their respective visible areas.

Window server 412 performs some additional "system-level" functions. For example, in order to support special user interface features such as "floating" windows, window server 412 maintains an ordering of window layers, and ensures that all windows are presented on the display in the appropriate order. Window server 412 also maintains an update area for each window which describes the part of the window that has been "damaged" due to window manipulations and window server 412 also notifies the view system 404 when one of the window update areas becomes non-empty. Window server 412 also maintains information about the actual display devices that make up the desktop. In order to do this, window server 412 interacts, as shown schematically by arrow 414, with two systems that are directly associated with the video display hardware. These devices include a hardware configuration system 416 and a hardware access system 418. As will hereinafter be described in detail, hardware access system 418 obtains specific hardware device characteristics and parameters by querying a small configuration memory that is included on each hardware device. The information gleaned from the configuration memories is passed from the hardware access system 418 to the window server 412 via the creation of specialized device driver objects and display device objects which are specific to a particular type of hardware. These specialized objects are then used to eventually create the screen buffer objects that are used by the view system to draw on the display devices. Commands and methods which are specific to specialized hardware are passed to the screen buffer objects and are then available in the view system for use by the application program.

The specialized device driver and display objects are also provided to the hardware configuration system 416 which contains configuration programs than can query and modify the display driver information to allow a user to change hardware specific information such as bit depth, monitoring, positioning and other hardware specific attributes of the displays.

Figure 5 is a more detailed schematic block diagram of the view system, the window server and the hardware access system and illustrates the interactions which are used to make specialized hardware capabilities available to an application program. As previously mentioned, view system 500 is used to create and manipulate windows on the screen area assigned to each application. For example, view system 500 manages "damage" within a document caused by view manipulation or invalidation and creates and destroys windows. The view system creates and destroys windows by creating an destroying corresponding window objects called "system window objects". More particularly, instantiating or creating a system window object from a pre-defined window class creates a corresponding window. Correspondingly, deleting a window object destroys the corresponding window.

The actual process involved in creating a window proceeds as follows: view system 500 creates a system window object using various parameters passed to it from the application program. These parameters may, for example, include the size or extent of the window, the front-to-back location of the window in relation to other windows, a task associated with the window that receives events, whether the window should be initially visible, and the kind of window.

View system 500 may create a plurality of windows for each application and the system window objects created for a particular application are illustrated in dotted box 502 with each system window object schematically represented by its corresponding extent shown as extents 504 and 506. The system window objects 502 are stored in the view system's allocated storage area. The window parameters used in creating each window object are sent to the window server 512 by means of a conventional data stream schematically illustrated as arrow 508. The operation and construction of such a data stream is well-known and will not be discussed in detail herein.

In response to the incoming window parameters, window server 512 actually creates the window area and returns the visible area of the newly-created window to the view system by means of a data stream (not shown). In order to create a window and to manipulate previously-created windows, window server 512 internally represents each window screen area as an "internal" window object. This internal window object is created, in part, using the window parameters sent from the view system 500. The internal windows corresponding to system windows 502 in the

view system are illustrated by window objects 514 and these window objects are schematically represented by their internal window "extents" shown as extents 516 and 518 which correspond to window objects 504 and 506, respectively. The internal window objects are stored in the storage area of the window server 512. The internal window objects actually perform manipulations on the windows in response to requests from the view system.

As will hereinafter be described in detail, the internal window objects 514 are created from pre-defined classes which, in general, correspond to generic display systems. In accordance with the present invention, windows which are to incorporate specialized hardware capabilities of display devices can be implemented by creating subclasses of the classes used to create the internal window objects. The correct subclasses which correspond with specialized hardware are then selected at system "power up" or during a reconfiguration by identifying the actual hardware devices and selecting the appropriate subclasses.

More particularly, for each display device that makes up the desktop, there exists a display device object contained in window server 512. For example, one of the display device objects in window server 512 is illustrated by box 522. As illustrated schematically by arrow 520, display device object 522 is used to create the internal window objects for windows which appear on the corresponding display device. By means of IO Access Manager 530, display device object 522 is created with, and knowledgeable of, any specialized hardware capabilities of the underlying hardware. Consequently, display device object 522 can then transfer these specialized hardware capabilities to the internal window objects 516 and 518 at the time that these window objects are created.

More particularly, for each video display device, display device IO Access Manager 530 (found in the hardware access system 528), creates a device driver object 534 as illustrated schematically by arrow 532. The device driver object 534 is specific to the type of hardware to which the display device IO Access Manager 530 is assigned, and, in order to create the device driver object 534, display device IO Access Manager 530 interacts directly with a particular piece of hardware 538. This interaction is shown schematically by arrow 535. Consequently, device driver object 534 can incorporate any specialized hardware capabilities of the underlying hardware device. Subsequently, device driver object 534 is used to create the display device object 522 as illustrated schematically by arrow 526.

In the process of creating internal window objects, such as internal window objects 516 and 518, display device object 522 also creates objects called graph device objects and screen device objects. These objects are used by view system 500 and application program 400 to draw graphics on the display device, and perform non-window operations such as sprite and cursor control. The graph device objects and

screen device objects are streamed back to the view system 500 as shown schematically by arrow 510. Arrow 510 represents a conventional data stream which may be the same data stream, or a different data stream than data stream 508.

By the creation chain as described above, the graph device objects and screen device objects created by the internal window objects also incorporate any specialized hardware capabilities in hardware 538.

Figure 6 shows a simplified class hierarchy diagram for classes and subclasses used to create an illustrative specialized display device driver object for hardware that has specialized video capabilities, such as motion video capabilities. In particular, the class (illustrated by box 612) used to construct the motion video device driver object is a subclass of two base classes, including a video device driver class 604 and a motion video class 608. This subclass relationship is represented by arrows 606 and 610, respectively. Therefore, when a motion video device driver object is created by the hardware access system (by invoking its constructor), the constructors of the underlying base objects will also be called in order to construct, and, optionally initialize, the base objects and include the functions and data present in the base objects.

The video device driver class 604 is, in turn, a subclass of a more generic device driver object 600. Display device capabilities which are available to the application program are determined by the class or subclass selected during the creation of the device driver object. For generic displays, class 604 is selected, whereas class 612 is selected when specific hardware capabilities are needed. As will hereinafter be described, the creation of a specialized video device driver object by subclassing class 604 to generate the motion video device driver 612 is accomplished by the display device IO Access Manager which identifies the actual hardware installed on the system and can therefore identify any hardware capabilities which are present. More particularly, any special capabilities of hardware 538 can be incorporated into a video device driver object created from class 604 by appropriately subclassing a generic video device driver class. This subclassing is done by the hardware device developer. The objects created from the classes are constructed dynamically by obtaining parameters from the hardware 538 and using those parameters to construct the appropriate object incorporating the necessary hardware capabilities.

A motion video device driver object created from class 612 also creates a motion video device handle 616 as illustrated schematically by arrow 614. The motion video device handle 616 is used by the window server to manipulate the motion video device driver.

Figure 7 shows a simplified class hierarchy diagram illustrating the classes and subclasses used in the creation of a specialized screen frame buffer object for use

by the view system and application programs. Figure 7 illustratively shows classes for the creation of a motion video screen frame buffer object corresponding to the motion video device driver object constructed from the classes illustrated in Figure 6.

The motion video screen frame buffer class 712 is a subclass of a more generic screen frame buffer class 708 as shown by arrow 710. The screen frame buffer class 708 is, in turn, a subclass of both a graph device class 700, which contains an interface to render graphic primitives, and a screen device class 702, which contains an interface to control non-window objects such as sprites and cursors, as illustrated by arrow 704 and 706. As previously mentioned, both the graph device class 700 and the screen device class 702 are effectively created by a motion video device driver object existing in the window server.

Figure 8 is an illustrative flow chart of the operations performed by the hardware access system (528, Figure 5) and the window server 512 in creating new device driver objects. The illustrative method starts in step 800 and proceeds to step 802 in which video hardware cards on the display or I/O bus are identified. This identification would normally be done by utilizing a software-driven identification process that starts during start up or boot of the computer system. Alternatively, this identification process may run during a hardware reconfiguration. This latter process checks each of the addresses associated with a particular I/O bus to locate video devices and other cards.

The routine illustrated in Figure 8 then proceeds to step 804 where a card "object" is created for each hardware card located on the bus. This creation is performed by instantiating a predefined card object using specific configuration information from each card. The actual configuration parameters of a specific card are obtained in one of several ways. One way is to read a small configuration ROM located on the hardware card. Such an arrangement is shown in Figure 9 in which two video hardware cards 904 and 910 are connected to the system I/O bus 900 as indicated schematically by arrows 902 and 908. Each of the video hardware cards 904 and 910 contains a small configuration ROM, such as ROMs 906 and 912, respectively. Information is stored in each configuration ROM in several mandatory information fields which include at least a format header block and a board resource list.

Alternatively, the configuration ROM may contain a card identifier code. This code can be used to access prestored device characteristic libraries containing particular card characteristics. The retrieved card characteristics can then be used to create the card object. If a video card is present on the system IO bus that does not have a configuration ROM, then a standard set of parameters can be substituted for the information which would normally be contained in the configuration ROM. In

any case, the created card object contains specific parameters that indicate the capabilities of the particular video hardware card.

The routine continues in step 806 in which each video card object creates a device driver object. Referring to Figure 5, the creation of the device driver object is illustrated schematically by arrow 532. The device driver object uses the specific hardware parameters from the card object to select an appropriate subclass for instantiation. As this subclass has been previously created by the hardware card developer, it includes the functionality necessary to exploit the particular board characteristics.

In step 808, the device driver object creates a display device object, an operation indicated schematically by arrow 526 in Figure 5. As with the device driver object, the appropriate subclass for the display device object is selected based on the specific hardware parameters. Next, in step 810, a display device handle is created by the display device object, which display device handle allows the display device object to be more easily more manipulated by the system as discussed in detail in the previously-mentioned co-pending patent application entitled "Object Oriented Video Framework System".

In step 812, display device handles are sent to the window server (this step is indicated schematically by arrow 520 in Figure 5). The display device handles are used to create the internal window objects and format the grafdevices and screen devices that are sent back to the view system. Since the display device handles correspond to the display device objects they select the proper subclasses to create the grafdevice and screen device objects which incorporate the specific hardware characteristics obtained from the video hardware cards. The routine then finishes in step 814.

Figure 10 illustrates a method by which hardware-dependent grafdevices and screen devices are incorporated into the view system. In particular, the routine starts in step 1000 and proceeds to step 1002. As previously mentioned, in order to create a given window on the display system, the view system instantiates a system window.

Next, in step 1004, the window descriptions corresponding to the newly created system window object are sent to the window server along with a request to register or create a new window, a process which is illustrated schematically by arrow 508 in Figure 5.

Next, in step 1006, within the window server, the window create request corresponding to the incoming window descriptions is dispatched to the correct display driver object. This dispatching is done by determining the display device on which the window will appear.

The window server then, in step 1008, requests that the display device object

create an internal window. A single internal window object is created unless the desired window spans several different monitors. In this latter case, an internal window cluster is created which is basically an internal window that has other internal windows included. A window cluster allows separate windows to be created on each respective display device.

In step 1010, the display device objects create grafdevices and screen devices which are returned to the view system in step 1012 (illustrated by arrow 510 and Figure 5). Since the internal windows are created from hardware specific subclasses, the window manipulations will be performed in the correct manner utilizing any hardware that is available. Further, since the grafdevice and screen device objects are created from the internal window objects, they will also be appropriate to the available hardware. Thus, drawing in a screen frame buffer created from these objects will also be appropriate to the existing hardware. The routine finishes in step 1014.

While the invention is described in terms of preferred embodiments in a specific system environment, those skilled in the art will recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the spirit and scope of the appended claims.

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

1. A computer system for controlling a display device having predefined characteristics to generate a display having a plurality of window areas displayed on a desktop background, each of the plurality of window areas displaying screen information generated by one of a plurality of application programs, the computer system comprising:
a processor controlled by the plurality of application programs to display screen information on the display device;
an operating system cooperating with the processor for controlling the display device;
apparatus associated with the display device for identifying the predefined characteristics; and
a hardware access system cooperating with the identifying apparatus and being responsive to identified predefined characteristics for creating a device driver object with specific commands and device characteristics for controlling the display device.

2. A computer system according to claim 1 wherein the hardware access system comprises predefined commands and data corresponding to a plurality of different display devices and the hardware access system is responsive to identified predefined characteristics for selecting a set of the predefined commands and data corresponding to the display device in order to create the device driver object.

3. A computer system according to claim 1 wherein the operating system comprises apparatus responsive to commands generated by one of the plurality of applications for creating a system window object which creates a specific window area on the display device and which comprises commands for drawing in a screen buffer corresponding to the specific window area.

4. A computer system according to claim 3 wherein the operating system further comprises a window server apparatus responsive to the system window object and to the device driver object for creating a screen device object which incorporates the specific commands and device characteristics for controlling the display device in the specific window area.

5. A computer system according to claim 4 wherein the operating system further comprises apparatus responsive to the screen device object for creating a screen buffer object comprising a screen buffer with specific commands and device characteristics for controlling the display device in the specific window area.

6. A computer system according to claim 3 wherein the computer system comprises a memory and wherein the one application program has exclusive access to an exclusive portion of the memory and wherein the system window object is created in the exclusive memory portion.

7. A computer system according to claim 6 wherein the memory further has a common portion which is accessible by each of the plurality of applications and the device driver object is created in the common portion.

8. A computer system according to claim 7 wherein the screen buffer object is created in the exclusive memory portion.

9. A computer system according to claim 1 wherein the identifying apparatus comprises a device memory located in the display device for storing the predefined characteristics.

10. A computer system according to claim 9 wherein the hardware access system comprises means for reading the predefined characteristics from the device memory.

11. A computer system for controlling a display device having predefined characteristics to generate a display having a plurality of window areas displayed on a desktop background, each of the plurality of window areas displaying screen information generated by one of a plurality of application programs, the computer system comprising:
a processor controlled by the plurality of application programs to display screen information on the display device;
an operating system cooperating with the processor for controlling the display device;
a device memory located in the display device for storing the predefined characteristics;
a hardware access system having apparatus for reading the device memory and having predefined device driver classes incorporating commands and characteristics for controlling the operation of a plurality of different display devices, the hardware access system being responsive to predefined characteristics read from the device memory for creating a device driver object from one of the predefined device driver classes;
apparatus responsive to commands generated by one of the plurality of applications for creating a system window object which creates a specific window area on the display device and which comprises commands for drawing in a screen buffer; and
a window server system responsive to the system window object and to the device driver object for creating a screen buffer.

12. A computer system according to claim 11 wherein the device memory contains a display device identification code and wherein the hardware access system comprises predefined commands and data corresponding to a plurality of different display devices and the hardware access system is responsive to a display device identification code read from the device memory for selecting a set of the predefined commands and data corresponding to the display device in order to create the device driver object.

13. A computer system according to claim 11 wherein the computer system comprises a memory and wherein the one application program has exclusive access to an exclusive portion of the memory and wherein the system window object is created in the exclusive memory portion.

14. A computer system according to claim 13 wherein the memory further has a common portion which is accessible by each of the plurality of applications and the device driver object is created in the common memory portion.

15. A computer system according to claim 14 wherein the screen buffer is created in the exclusive memory portion.

16. A computer system for controlling a display device having predefined characteristics to generate a display having a plurality of window areas displayed on a desktop background, each of the plurality of window areas displaying screen information generated by one of a plurality of application programs, the computer system comprising:
a processor controlled by the plurality of application programs to display screen information on the display device;
a memory having a common memory portion and a plurality of exclusive memory portions, one of the plurality of exclusive memory portions being accessible by the one application program;
an operating system cooperating with the processor for controlling the display device;
a device memory located in the display device for storing the predefined characteristics;
a hardware access system having predefined device driver classes incorporating commands and characteristics for controlling the operation of a plurality of different display devices, the hardware access system being responsive to predefined characteristics read from the device memory for creating a device driver object in the common memory portion from one of the predefined device driver classes; and
apparatus responsive to parameters generated by one of the plurality of applications for creating a system window object in the one exclusive memory portion, which system window object creates a specific window area on the display device and which system window object comprises commands for drawing in a screen buffer corresponding to the specific window area.

17. A computer system according to claim 16 wherein the computer system further comprises apparatus for creating a window server object in the common memory portion and a first data stream object for transferring the parameters from the system window object to the window server object.

18. A computer system according to claim 17 wherein the window server object comprises apparatus responsive to the parameters transferred from the system window object for creating an internal window object wherein the internal window object is responsive to the window parameters and to the display device driver for creating the screen buffer.

19. A computer system according to claim 18 wherein the computer system further comprises a second data stream object for transferring the screen buffer from the window server object to the system window object.

20. A method for controlling a display device in a computer system, the display device having predefined characteristics in order to generate a display having a plurality of window areas displayed on a desktop background, each of the plurality of window areas displaying screen information generated by one of a plurality of application programs, the method comprising the steps of:
A. examining the display device to identify the predefined characteristics; and
B. using the identified predefined characteristics to create a device driver object with specific commands and device characteristics for controlling the display device.

21. A method according to claim 20 wherein step A comprises the step of:
A1. reading a display device identification code from the display device and step B comprises the steps of:
B1. compiling a set of predefined commands and data corresponding to a plurality of different display devices;
B2. selecting a set of the predefined commands and data based on the display device identification code read from the display device; and
B3. creating the device driver object using the set selected in step B2.

22. A method according to claim 20 further comprising the steps of:
C. receiving parameters generated by one of the plurality of applications; and
D. using the parameters received in step C for creating a system window object which creates a specific window area on the display device and which comprises commands for drawing in a screen buffer corresponding to the specific window area.

23. A method according to claim 22 further comprising the step of creating a screen device object which incorporates the specific commands and device characteristics for controlling the display device in the specific window area using the system window object and the device driver object.

24. A method according to claim 23 further comprising the step of creating a screen buffer object comprising a screen buffer with specific commands and device characteristics for controlling the display device in the specific window area using the screen device object.

25. A method according to claim 24 wherein the computer system comprises a memory and wherein the one application program has exclusive access to an exclusive portion of the memory and including the step of creating the system window object in the exclusive memory portion.

26. A method according to claim 25 wherein the memory further has a common portion which is accessible by each of the plurality of applications and including the step of creating the device driver object in the common memory portion.

27. A method according to claim 26 including the step of creating the screen buffer object in the exclusive memory portion.

28. A method according to claim 20 including the step of reading the predefined characteristics from a device memory located in the display device.

29. A method for controlling a display device in a computer system, the display device having predefined characteristics to generate a display having a plurality of window areas displayed on a desktop background, each of the plurality of window areas displaying screen information generated by one of a plurality of application programs, the computer system having a memory with a common memory portion and a plurality of exclusive memory portions, one of the plurality of exclusive memory portions being accessible by the one application program, the method comprising the steps of:

A. storing the predefined characteristics in a device memory located in the display device;

B. storing a set of predefined device driver classes incorporating commands and characteristics for controlling the operation of a plurality of different display devices in the common memory portion;

C. reading the predefined characteristics from the device memory;

D. creating a device driver object in the common memory portion using predefined characteristics read from the device memory from one of the predefined device driver classes;

E. receiving window parameters from one of the plurality of applications; and

F. creating a system window object in the one exclusive memory portion, which system window object creates a specific window area on the display device and which system window object comprises commands for drawing in a screen buffer corresponding to the specific window area.

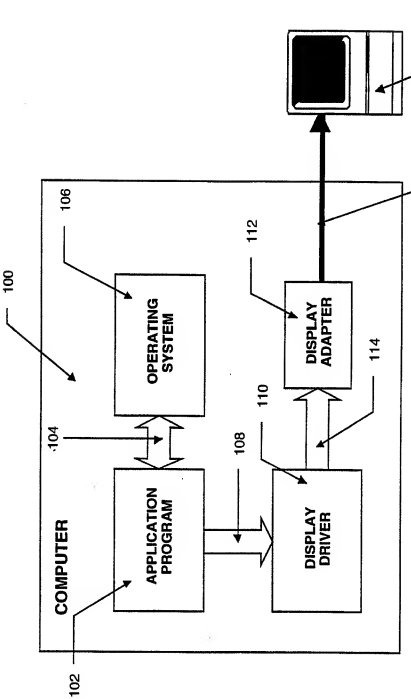


FIGURE 1 **(Prior Art)**

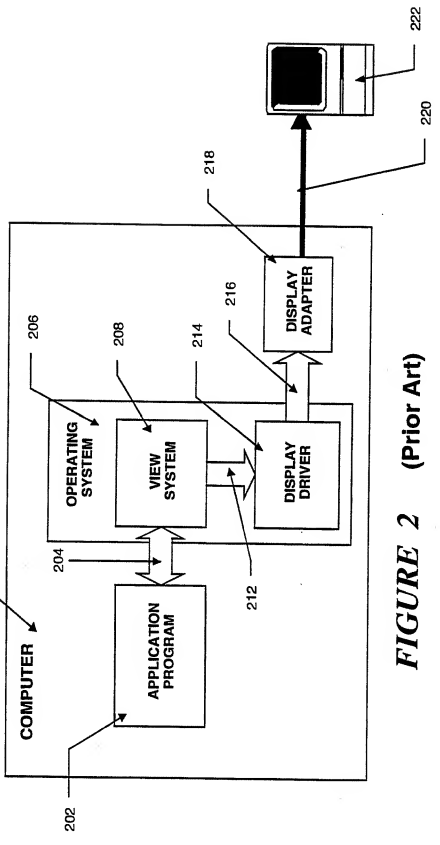


FIGURE 2 (Prior Art)

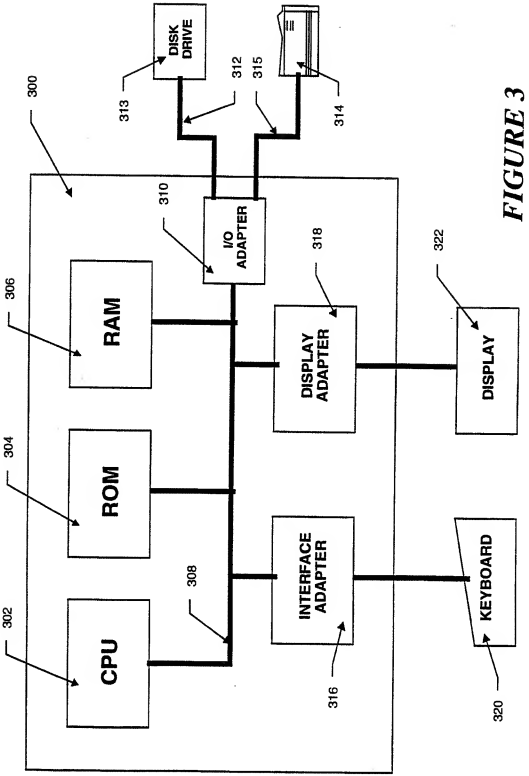

300
302
CPU
304
ROM
306
RAM
308
310
I/O ADAPTER
312
313
DISK DRIVE
314
315
316
INTERFACE ADAPTER
318
DISPLAY ADAPTER
320
KEYBOARD
322
DISPLAY

FIGURE 3

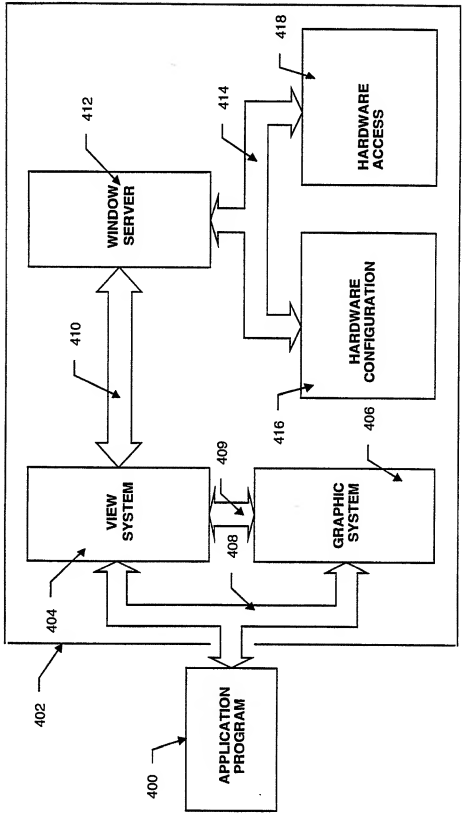

400
APPLICATION PROGRAM
402
404
VIEW SYSTEM
408
409
GRAPHIC SYSTEM
406
410
WINDOW SERVER
412
414
HARDWARE CONFIGURATION
416
HARDWARE ACCESS
418

FIGURE 4

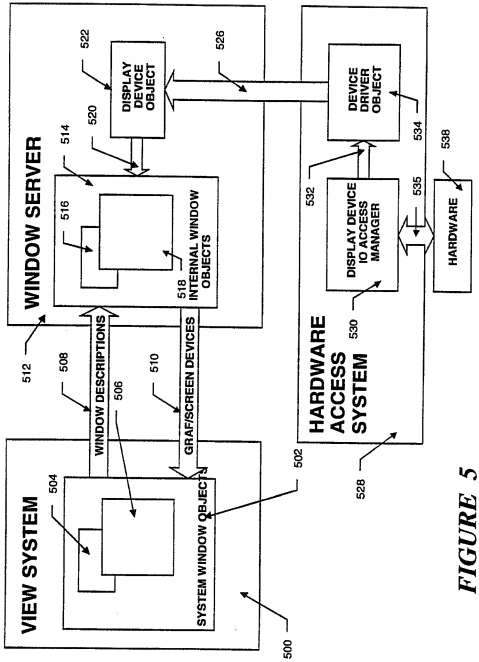

VIEW SYSTEM
504
SYSTEM WINDOW OBJECTS
500
WINDOW DESCRIPTIONS
506
508
GRAF/SCREEN DEVICES
510
502
WINDOW SERVER
512
514
516
518
INTERNAL WINDOW OBJECTS
520
522
DISPLAY DEVICE OBJECT
526
HARDWARE ACCESS SYSTEM
528
530
DISPLAY DEVICE IO ACCESS MANAGER
532
DEVICE DRIVER OBJECT
534
535
HARDWARE
538

FIGURE 5

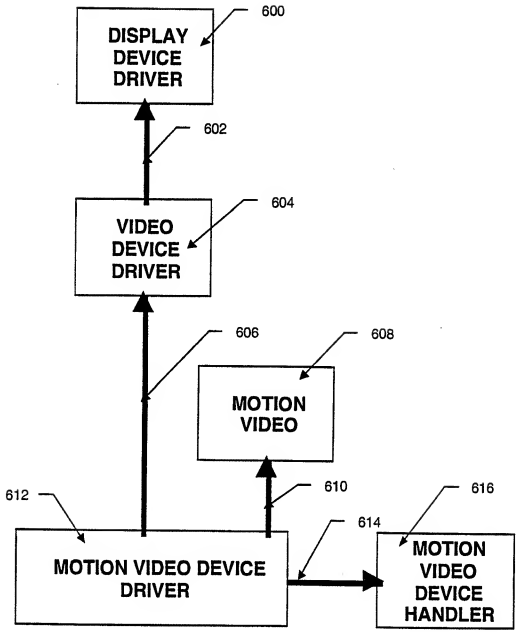

DISPLAY DEVICE DRIVER
600
602
VIDEO DEVICE DRIVER
604
606
608
MOTION VIDEO
610
612
614
616
MOTION VIDEO DEVICE DRIVER
MOTION VIDEO DEVICE HANDLER

FIGURE 6

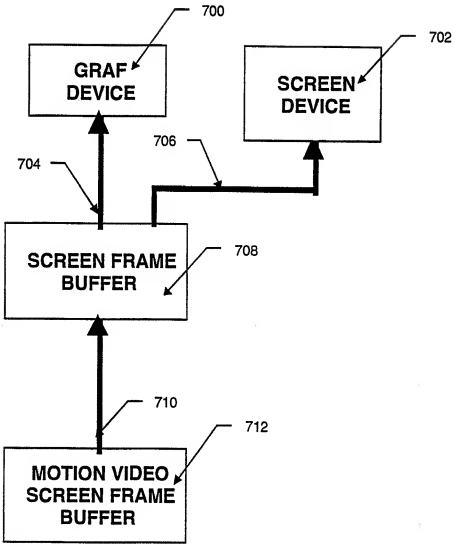

700
GRAF DEVICE
702
SCREEN DEVICE
706
704
SCREEN FRAME BUFFER
708
710
712
MOTION VIDEO SCREEN FRAME BUFFER

FIGURE 7

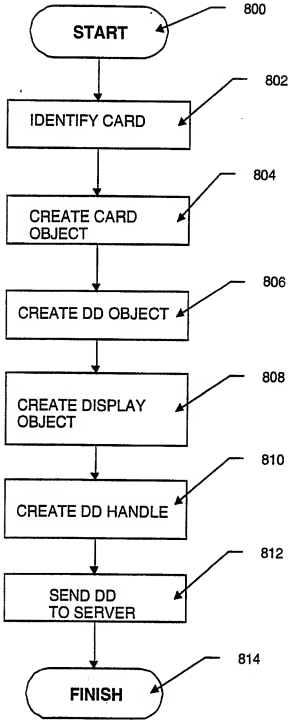

START
800
IDENTIFY CARD
802
CREATE CARD OBJECT
804
CREATE DD OBJECT
806
CREATE DISPLAY OBJECT
808
CREATE DD HANDLE
810
SEND DD TO SERVER
812
FINISH
814

FIGURE 8

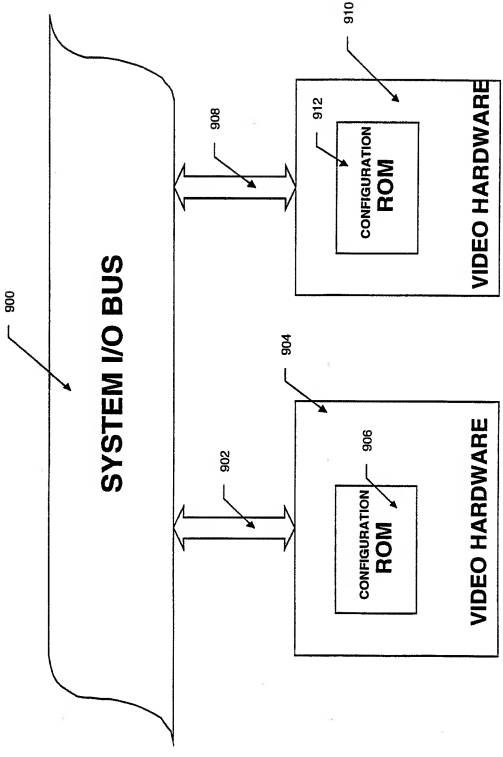

900
SYSTEM I/O BUS
902
904
VIDEO HARDWARE
CONFIGURATION ROM
906
908
910
VIDEO HARDWARE
912
CONFIGURATION ROM

FIGURE 9

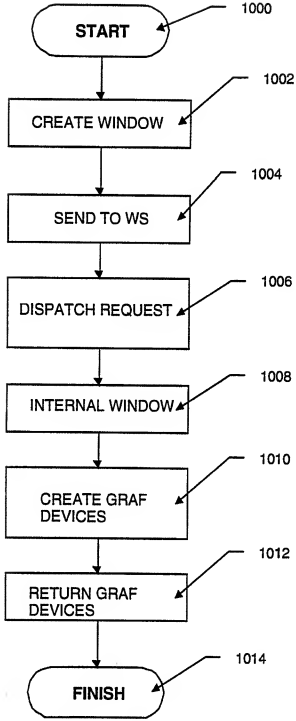

START
1000
CREATE WINDOW
1002
SEND TO WS
1004
DISPATCH REQUEST
1006
INTERNAL WINDOW
1008
CREATE GRAF DEVICES
1010
RETURN GRAF DEVICES
1012
FINISH
1014

FIGURE 10

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/445 G06F3/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	P. KOUGIOURIS 'A device management framework for an object oriented operating system' May 1991 , UNIV. OF ILLINOIS , URBANA, US Report no. UIUCDCS-R-91-1707 Chapter 5, Choices Device Management see page 27 - page 43	1,2
Y	---	3-5,9,10
Y	'Inside Macintosh, Volumes I, II and III' 1985 , APPLE COMPUTER INC. , CUPERTINO, US Vol. I, Chapter 9, The Window Manager see page I-267 - page I-277 Vol. II, Chapter 6, The Device Manager see page II-173 - page II-195	3-5,9,10
A	---	2,6
	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 July 1994	20.07.94

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Kingma, Y

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	'Inside Macintosh, Volume V' 1986 , APPLE COMPUTER INC. , CUPERTINO, US Chapter 5, Graphics Devices see page V-117 - page V-131 Chapter 19, The Start Manager see page V-347 - page V-356 Chapter 23, The Device Manager see page V-421 - page V-428 Chapter 24, The Slot Manager see page V-435 - page V-457	2
A	EP,A,0 398 644 (INTERNATIONAL BUSINESS MACHINES CORP.) 22 November 1990 see claims 1-4,6-9	
A	GB,A,2 248 130 (YOKOGAWA ELECTRIC CORP.) 25 March 1992 see page 1, line 1 - page 2, line 21	5,6,8

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0398644	22-11-90	CA-A,C	2016396	15-11-90
		JP-A-	3006615	14-01-91
GB-A-2248130	25-03-92	JP-A-	1251094	06-10-89
		JP-A-	2114293	26-04-90
		JP-A-	2114294	26-04-90
		JP-A-	2114295	26-04-90
		DE-A-	3908507	19-10-89
		GB-A,B	2217155	18-10-89
		GB-A,B	2248157	25-03-92
		GB-A,B	2248158	25-03-92
		NL-A-	8900797	16-10-89
		US-A-	5065343	12-11-91